

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.

No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.

See full “License and Conditions of Use” notice for details.

In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Author: Carlos Daniel Borrego Romero

Annex C. Reference Architecture for Institutional Integration, Python Implementation, and Explanatory Power of the Model

This annex proposes a general reference architecture for institutional integration of the crustal deformation and log-log convergence model and includes Python code intended to facilitate adoption in observatories, research centers, and civil-protection environments. It also develops the explanatory power of the model by showing how the framework links deformation, catalog evolution, auxiliary anomalies, and critical-time convergence into a coherent interpretation of rupture preparation. The architecture is designed to be realistic with current institutional technology, modular enough for phased deployment, and compatible with AI-assisted monitoring pipelines already emerging in geophysics. [web:299][web:301][web:302][web:304][web:306][file:242]

1. General reference architecture

A practical institutional architecture for the model can be organized into six layers.

1. **Acquisition layer:** continuous ingestion of GNSS, InSAR, seismic catalog, waveform-derived, tilt, strain, gas, hydrological, and thermal data depending on the hazard domain. Modern observatories already run digital acquisition pipelines and Python-compatible data services for many of these streams. [web:301][web:302][web:304][web:306]
2. **Quality-control and harmonization layer:** time alignment, detrending, denoising, missing-data handling, station-quality screening, and transformation into common analysis windows. Automated seismic QC frameworks and cloud monitoring workflows show that this layer is already feasible at operational scale. [web:299][web:302]
3. **Feature-engineering layer:** conversion of raw observations into the model components $X_1(t)$, $X_2(t)$, $X_3(t)$, and $X_4(t)$, including normalization and bounded scoring. Python-based geophysical software ecosystems already support modular signal processing and inversion-oriented workflows that fit this step well. [web:294][web:295][web:304]

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.

No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.

See full “License and Conditions of Use” notice for details.

In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Author: Carlos Daniel Borrego Romero

4. **Model-execution layer:** rolling computation of the normalized precursor index $I^*(t)$, moving-window estimation of $\hat{t}_c(t)$, false-positive screening, and alert-state evaluation. [file:242]
5. **AI-assisted support layer:** anomaly ranking, adaptive calibration, multimodal fusion, pattern discrimination, and analyst prioritization. Scalable monitoring workflows such as QuakeFlow show that real-time and batch AI-supported geophysical pipelines are already technically achievable. [web:302][web:306][web:308]
6. **Institutional delivery layer:** dashboards, analyst review interfaces, scenario escalation pathways, archived reports, and interfaces to civil-protection decision chains. [web:302][file:242]

The reference flow is therefore: **acquisition** → **harmonization** → **feature engineering** → **model execution** → **AI-assisted support** → **analyst-facing delivery**. This structure enables observatories to adopt the model incrementally rather than through full system replacement. [web:299][web:301][web:302][file:242]

2. Functional design principles

Several design principles are essential for institutional deployment. First, the architecture should remain modular, allowing different institutions to activate only the components for which they have reliable coverage. A tectonic observatory may begin with geodetic and seismic inputs, while a volcanic observatory may incorporate gas, thermal, and hydrothermal signals as soon as those streams are stable. [web:252][web:255][file:242]

Second, the pipeline should remain transparent and physically interpretable. The model gains much of its value from linking each stage of the computation to a recognizable geophysical process, so every derived feature should be traceable to its source signal, preprocessing step, and normalization rule. [file:242]

Third, AI should be integrated as a support layer rather than as an opaque replacement for geophysical reasoning. Current operational evidence supports using AI for fusion, screening, and prioritization, while preserving the physics-based core of the forecasting logic and the authority of human analysts. [web:302][web:306][web:308]

3. Python reference implementation

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.

No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.

See full “License and Conditions of Use” notice for details.

In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Author: Carlos Daniel Borrego Romero

The following Python code provides a modular institutional scaffold. It is designed to illustrate how acquisition-ready records, normalization, feature generation, model fitting, and structured reporting can be organized inside a single pipeline. The code is intentionally explicit so that observatories can adapt individual components without changing the overall architecture.

[web:294][web:299][web:302][file:242]

```
from dataclasses import dataclass, field
from typing import Dict, List, Any, Optional
import math
from statistics import mean

@dataclass
class MonitoringRecord:
    region_id: str
    hazard_type: str
    geodetic: Dict[str, Any] = field(default_factory=dict)
    seismic: Dict[str, Any] = field(default_factory=dict)
    auxiliary: Dict[str, Any] = field(default_factory=dict)
    metadata: Dict[str, Any] = field(default_factory=dict)

class AcquisitionLayer:
    def from_observatory_bundle(self, bundle: Dict[str, Any]) -> MonitoringRecord:
        return MonitoringRecord(
            region_id=bundle.get("region_id", "unknown"),
            hazard_type=bundle.get("hazard_type", "tectonic"),
            geodetic=bundle.get("geodetic", {}),
            seismic=bundle.get("seismic", {}),
            auxiliary=bundle.get("auxiliary", {}),
            metadata=bundle.get("metadata", {}),
        )

class Normalization:
    @staticmethod
    def minmax(value: float, lower: float, upper: float) -> float:
        if value > upper:
            raise ValueError("upper must be greater than lower")
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.

No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.

See full “License and Conditions of Use” notice for details.

In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Author: Carlos Daniel Borrego Romero

```
x = (value - lower) / (upper - lower)
return max(0.0, min(1.0, x))

@staticmethod
def logistic(z: float) -> float:
    return 1.0 / (1.0 + math.exp(-z))

class FeatureEngineeringLayer:
    def __init__(self, config: Dict[str, Any]):
        self.config = config

    def compute_x1_velocity(self, geodetic: Dict[str, Any]) -> float:
        v = float(geodetic.get("velocity_anomaly", 0.0))
        cfg = self.config["x1_velocity"]
        return Normalization.minmax(v, cfg["lower"], cfg["upper"])

    def compute_x2_acceleration(self, geodetic: Dict[str, Any]) -> float:
        a = float(geodetic.get("acceleration_anomaly", 0.0))
        cfg = self.config["x2_acceleration"]
        return Normalization.minmax(a, cfg["lower"], cfg["upper"])

    def compute_x3_auxiliary(self, auxiliary: Dict[str, Any]) -> float:
        vals = auxiliary.get("normalized_anomaly_scores", [])
        if not vals:
            return 0.0
        return max(0.0, min(1.0, mean(vals)))

    def compute_x4_seismic(self, seismic: Dict[str, Any]) -> float:
        event_rate = float(seismic.get("event_rate_anomaly", 0.0))
        b_value_inverse = float(seismic.get("b_inverse_anomaly", 0.0))
        swarm = float(seismic.get("swarm_density", 0.0))
        cfg1 = self.config["x4_event_rate"]
        cfg2 = self.config["x4_b_inverse"]
        cfg3 = self.config["x4_swarm"]
        s1 = Normalization.minmax(event_rate, cfg1["lower"], cfg1["upper"])
        s2 = Normalization.minmax(b_value_inverse, cfg2["lower"], cfg2["upper"])
        s3 = Normalization.minmax(swarm, cfg3["lower"], cfg3["upper"])
        return (s1 + s2 + s3) / 3.0
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.

No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.

See full “License and Conditions of Use” notice for details.

In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Author: Carlos Daniel Borrego Romero

```
class ForecastModel:
    def __init__(self, weights: Dict[str, float], epsilon: float = 1e-6):
        self.weights = weights
        self.epsilon = epsilon

    def compute_index(self, x1: float, x2: float, x3: float, x4: float) -> float:
        return (
            self.weights["x1"] * x1 +
            self.weights["x2"] * x2 +
            self.weights["x3"] * x3 +
            self.weights["x4"] * x4
        )

    def estimate_tc_proxy(self, index_history: List[float], dt: float) -> Optional[float]:
        if len(index_history) < 3:
            return None
        recent = index_history[-3:]
        growth = recent[-1] - recent[0]
        if growth <= 0:
            return None
        slope = growth / (2 * dt + self.epsilon)
        if slope <= 0:
            return None
        remaining = max(0.0, 1.0 - recent[-1])
        return remaining / (slope + self.epsilon)

    def convergence_score(self, tc_estimates: List[float]) -> Optional[float]:
        if len(tc_estimates) < 3:
            return None
        window = tc_estimates[-3:]
        mu = mean(window)
        if mu <= 0:
            return None
        dispersion = sum(abs(x - mu) for x in window) / len(window)
        return max(0.0, 1.0 - dispersion / (mu + self.epsilon))

class AIAssistLayer:
    def anomaly_priority(self, x1: float, x2: float, x3: float, x4: float, convergence: Optional[float]) -> float:
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.

No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.

See full “License and Conditions of Use” notice for details.

In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Author: Carlos Daniel Borrego Romero

```
base = 0.3 * x1 + 0.3 * x2 + 0.15 * x3 + 0.25 * x4
if convergence is not None:
    base = 0.7 * base + 0.3 * convergence
return max(0.0, min(1.0, base))

class ReportingLayer:
    def build_report(self, record: MonitoringRecord, features: Dict[str, float], index_value: float,
                    tc_proxy: Optional[float], convergence: Optional[float], priority: float) -> Dict[str, Any]:
        status = "monitor"
        if convergence is not None and convergence > 0.85 and priority > 0.75:
            status = "enhanced analyst review"
        return {
            "region_id": record.region_id,
            "hazard_type": record.hazard_type,
            "features": features,
            "normalized_index": round(index_value, 4),
            "tc_proxy": None if tc_proxy is None else round(tc_proxy, 4),
            "convergence_score": None if convergence is None else round(convergence, 4),
            "priority_score": round(priority, 4),
            "status": status,
        }

def run_pipeline(bundle: Dict[str, Any], config: Dict[str, Any], weights: Dict[str, float],
                index_history: List[float], tc_estimates: List[float], dt: float = 1.0) -> Dict[str, Any]:
    record = AcquisitionLayer().from_observatory_bundle(bundle)
    fe = FeatureEngineeringLayer(config)

    x1 = fe.compute_x1_velocity(record.geodetic)
    x2 = fe.compute_x2_acceleration(record.geodetic)
    x3 = fe.compute_x3_auxiliary(record.auxiliary)
    x4 = fe.compute_x4_seismic(record.seismic)

    model = ForecastModel(weights=weights)
    index_value = model.compute_index(x1, x2, x3, x4)
    index_history = index_history + [index_value]
    tc_proxy = model.estimate_tc_proxy(index_history, dt=dt)

    if tc_proxy is not None:
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.
Author: Carlos Daniel Borrego Romero

```
tc_estimates = tc_estimates + [tc_proxy]
convergence = model.convergence_score(tc_estimates)

ai = AIAssistLayer()
priority = ai.anomaly_priority(x1, x2, x3, x4, convergence)

return ReportingLayer().build_report(
    record=record,
    features={"x1_velocity": x1, "x2_acceleration": x2, "x3_auxiliary": x3, "x4_seismic": x4},
    index_value=index_value,
    tc_proxy=tc_proxy,
    convergence=convergence,
    priority=priority,
)
```

This reference implementation is not a final operational system, but it expresses the essential institutional logic of the model: structured data ingestion, bounded feature construction, multicomponent precursor synthesis, rolling criticality tracking, AI-assisted prioritization, and analyst-facing output. It is well aligned with current Python-based geophysical ecosystems, which already support modular processing, data-quality control, and scalable monitoring workflows. [web:294][web:299][web:301][web:302][web:304]

4. Explanatory power of the model

The explanatory power of the model lies in how it transforms heterogeneous precursor signals into a coherent interpretation of system evolution. Rather than treating deformation, seismicity, and auxiliary anomalies as disconnected warnings, the framework interprets them as interacting manifestations of a common transition toward failure. This produces an explanatory narrative that is stronger than isolated threshold exceedance alone. [file:242]

In tectonic settings, the model explains why rupture risk may increase even before a major catalog event occurs: deformation velocity captures the immediate rate of crustal change, deformation acceleration captures the intensification of that rate, the auxiliary anomaly component reflects coupled fluid or environmental perturbation, and the seismic component captures the microstructural organization of the fault zone. When these four dimensions move coherently, the model interprets the system not as merely noisy, but as dynamically converging toward a critical regime. [file:242]

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.
Author: Carlos Daniel Borrego Romero

In volcanic settings, the same explanatory logic is equally powerful. Surface uplift, acceleration of deformation, hydrothermal or gas anomalies, and evolving volcanic seismicity are not treated as separate monitoring dashboards, but as parts of one evolving instability structure. This is particularly valuable because volcanic unrest often unfolds through interacting processes rather than through a single dominant signal. [web:252][web:255][file:242]

The model also improves interpretability in relation to time. Many monitoring approaches detect anomalies but do not naturally connect them to an estimated candidate failure time. By introducing a rolling convergence logic around $\hat{t}_c$, the framework adds temporal structure to the explanatory process: it distinguishes between elevated unrest, accelerating instability, unstable convergence, and stabilized candidate criticality. [file:242]

This explanatory structure has operational value because it supports communication. Analysts can describe not only that a region is anomalous, but why it is anomalous, which dimensions are driving the change, how coherent those dimensions are, and whether the convergence pattern is strengthening or weakening. In institutional settings, this is often as important as raw predictive output. [file:242][web:302]

5. Concluding integration paragraph

Annex C shows that the model can be translated into a realistic institutional architecture using technologies that are already available in observatories and research centers: Python-based processing, automated QC layers, modular feature engineering, rolling execution, AI-assisted analyst support, and dashboard-oriented delivery. At the same time, the model offers significant explanatory power because it unifies deformation, seismic, and auxiliary signals within one critical-transition logic that remains physically interpretable. [web:299][web:301][web:302][web:304][web:306][file:242]

This combination of implementability and explanatory coherence is one of the model's strongest advantages. It allows institutions to treat the framework not only as a forecasting experiment, but also as an interpretive bridge between heterogeneous observables and operational decision support. [file:242]

LICENSE AND CONDITIONS OF USE.

The content is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0) license:
<https://creativecommons.org/licenses/by-nc-sa/4.0/>

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.

No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.

See full "License and Conditions of Use" notice for details.

In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Author: Carlos Daniel Borrego Romero

By using this content and model, you agree to:

Provide appropriate credit, include a link to the above license, and indicate if changes were made.

Not use the material or any derivative works for commercial purposes.

Distribute any adaptations or derivative works only under the same CC BY-NC-SA 4.0 license.

Not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

Data logging and trade secrets policy.

The design, implementation, and deployment of this model, as well as any derivatives, do not authorize the collection, storage, or exploitation of IP addresses, unique device identifiers, or other technical traceability data for commercial profiling, targeted advertising, or any other economic exploitation.

Any system implementing this model must limit the logging and retention of IP addresses and other identifiable data strictly to what is necessary to:

(a) maintain technical security and service integrity; and

(b) comply with applicable legal obligations (for example, any minimum log retention required by law).

It is prohibited to use this model or its derivatives to:

capture, store, or exploit trade secrets, proprietary know-how, or confidential information of users or third parties for any purpose other than providing the technical service described in this documentation; or train or improve closed, commercial systems using data that contains confidential information of third parties without their prior, explicit, written consent.

---+EOD+---